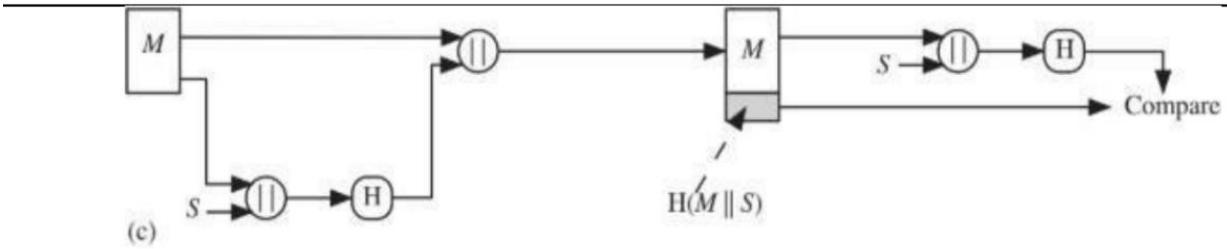
PRACTICAL: 7

AIM:

Refer to the attached figure here. Bob is preparing to send message to Alice. Bob and Alice both secretly computes the code(s) without sharing on any communication channel. Suggest key exchange algorithm to Bob and Alice for securely exchange information without sharing actual key. Once they form secret code, Bob applies SHA256 hash algorithm on original message (M) plus code (s) and send hash of original message and code (M||s) to Alice. Alice will receive bundle of H(M||s) and first append code (s) with received message (M) and produce hash of the message (H) that compare with H(M||s) to make sure that message is not altered by any attackers.



The task to perform:

1. Use some key exchange algorithm to calculate the value of s (secret code) which must be unique at the sender and receiver side.

2. Implementation can be done using any programming language such as Java programming or python programming.

3. Apply SHA256 on t h e message and secret code and display it on the output screen. Verify the hash value at the receiver end.

THEORY:

Key exchange algorithms allow two parties to securely establish a shared secret without directly transmitting the key. One of the most widely used key exchange algorithms is Diffie-Hellman (DH) Key Exchange, which enables Bob and Alice to generate a unique secret key (s) without exposure.

In this practical, Bob and Alice independently generate the secret code (s) using a key exchange algorithm. Once established, Bob applies the SHA-256 hashing algorithm to the concatenated message and secret code (M||s) and transmits the hash to Alice. Alice then verifies the integrity of the received message by computing H(M||s) and comparing it with Bob's transmitted hash.

SHA-256 is a cryptographic hash function that ensures data integrity and prevents unauthorized modifications. The received hash is validated to check whether the message was altered during transmission.

CODE:

```
import java.util.*;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

public class trialfile {

   public static final BigInteger P = new BigInteger("23");
   public static final BigInteger G = new BigInteger("5");

   public static void main(String[] args) throws NoSuchAlgorithmException {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Bob, enter your private key: ");
      BigInteger bobPrivateKey = scanner.nextBigInteger();

      System.out.print("Alice, enter your private key: ");
      BigInteger alicePrivateKey = scanner.nextBigInteger();

      BigInteger bobPublicKey = G.modPow(bobPrivateKey, P);
      BigInteger alicePublicKey = G.modPow(alicePrivateKey, P);

      BigInteger bobSharedSecret = alicePublicKey.modPow(bobPrivateKey, P);
      BigInteger aliceSharedSecret = bobPublicKey.modPow(alicePrivateKey, P);

      if (bobSharedSecret.equals(aliceSharedSecret)) {
         System.out.println("Shared secret key established: " + bobSharedSecret);
      } else {
         System.out.println("Error in key exchange!");
         return;
      }

      System.out.print("Bob, enter the message: ");
      scanner.nextLine();
      String message = scanner.nextLine();

      String messageWithSecret = message + bobSharedSecret;
      String hash = sha256(messageWithSecret);

      System.out.println("Bob sends:");
      System.out.println("Message: " + message);
      System.out.println("Hash: " + hash);
```

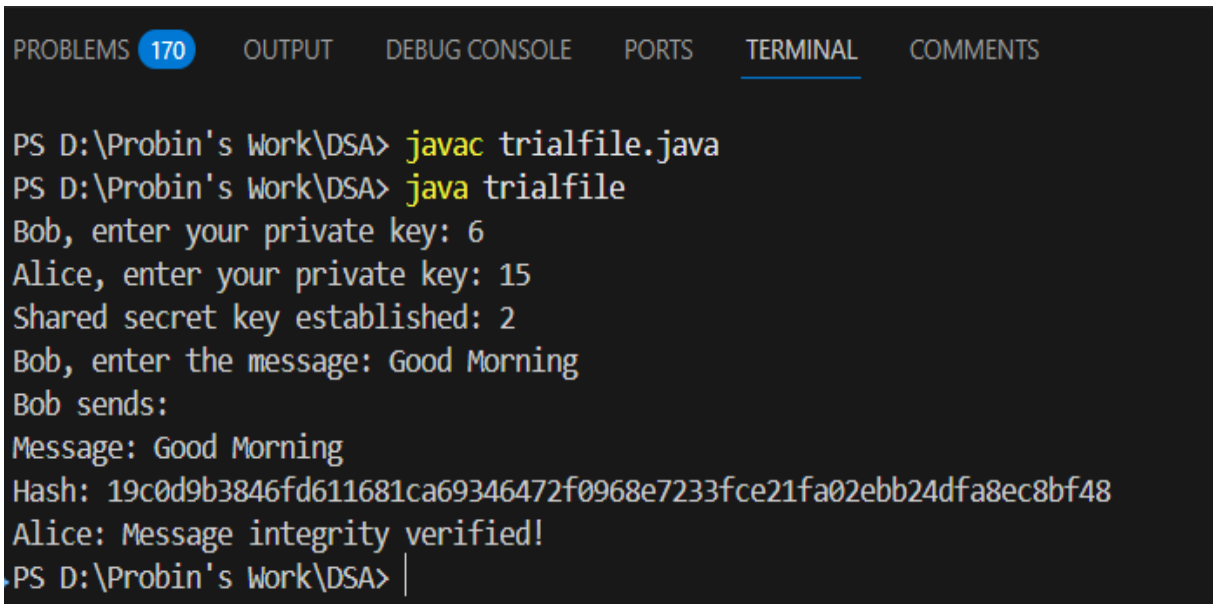
```
        String receivedMessage = message;
        String verifyHash = sha256(receivedMessage + aliceSharedSecret);

        if (verifyHash.equals(hash)) {
            System.out.println("Alice: Message integrity verified!");
        } else {
            System.out.println("Alice: Message was altered!");
        }

        scanner.close();
    }

    // SHA-256 Hash Function
    public static String sha256(String input) throws NoSuchAlgorithmException {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        byte[] hashBytes = digest.digest(input.getBytes());
        StringBuilder hexString = new StringBuilder();
        for (byte b : hashBytes) {
            hexString.append(String.format("%02x", b));
        }
        return hexString.toString();
    }
}
```

OUTPUT:



LATEST APPLICATIONS:

1. **Secure Online Transactions** – Used in digital banking and payment systems to verify transaction integrity. This prevents unauthorized modifications to financial data, ensuring safe and tamper-proof transactions.

2. **Digital Signatures** – Ensures data authenticity and prevents tampering in emails and legal documents. Businesses and government agencies use this technique to verify the legitimacy of digital contracts and sensitive communications.

3. **Blockchain Security** – Hashing algorithms like SHA-256 secure transactions in cryptocurrencies like Bitcoin. This ensures that once a block is added to the blockchain, its contents cannot be altered, making transactions immutable.

LEARNING OUTCOME:

1. Understand Key Exchange Algorithms– I understood how Diffie-Hellman securely establishes a shared secret between two parties which is essential for designing secure communication systems in real-world applications.

2. Implement SHA-256 for Message Integrity– I applied cryptographic hashing to verify that data remains unaltered during transmission which helps in securing sensitive data, ensuring that the transmitted information remains intact and unchanged.

3. Enhance Cybersecurity Skills– Gained insights into real-world secure communication techniques by implementing encryption and hashing methods. This helped me understand how to protect data from cyber threats and unauthorized modifications.

REFERENCES:

1. https://www.geeksforgeeks.org/implementation-diffie-hellman-algorithm/
2. https://www.techtarget.com/searchsecurity/definition/Diffie-Hellman-key-exchange
3. https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm